Class Project 1

Introduction to Big Data & Analytics

Kalyan Kollepara

Prashanth Kumar Manji

Professor: Steve Kaisler

1. Dataset: Introduction

Our goal for this project was to load and simplify the graph in the most simple yet efficient way possible. We tackled this problem by attempting to use the least amount of code possible for our functions. As we know from working with R in the past, it is possible to go down a "rabbit hole" where you spend many lines of code trying to make functions work. In reality, many simplifications in R can be done with a few lines of code using matrix transformations and other functional methods.

Loading the data from the text file was relatively simple. We converted the text file into a table (after skipping the first couple commented lines) and after being in table form we were able to use the data more efficiently. We were able to convert the table into a matrix and then to a dataframe to be used by iGraph.

To get the essence of the graph we have used some functions to identify the edges, vertices, edge density and degree of nodes. We used the Simplify function to simplify the data. But, the graph does not have any loops or multiple edges. We took samples of nodes in starting, middle and at the end independently and examined them by using plots of the graph to study how they looked different from each other. This helped us to understand how the nodes in a graph are interconnected.

2. Install the igraph package from one of the CRAN mirrors. Determine how to create a graph and plot. Show the plot in your report.

Installation of igraph was done by doing a simple library(igraph) call. In order to get the .txt file in table form for better R programming, we used the read.table() function. There needed to be certain specifications made in order to get the proper contingency table. For example, the .txt file contained lines that were not useful and therefore had to be skipped. Figure 1 shows the table after conversion to a data frame. The conversion to a data frame was done in order to turn the data into a graph via igraph. We will refer to this graph as Graph g and was created using the graph.data.frame() function as seen in Figure 1.4. Figure 1.2 shows the edges are labeled with the relationship each node has to each other. Figure 1.4 shows the full graph and we found it almost impossible to make it properly formatted using RStudio.In Figure 1.3, we plotted the graph for the last 100 nodes inorder to avoid overlapping and better understanding. Figure 1.4 gives the code that was used to read the data, turn it into a graph, store the graph, and then plot it.

	FromNodeId	ToNodeId
1	3466	937
2	3466	5233
3	3466	8579
4	3466	10310
5	3466	15931
6	3466	17038
7	3466	18720
8	3466	19607
9	10310	1854
10	10310	3466
11	10310	4583
12	10310	5233
13	10310	9572
14	10310	10841
15	10310	13056
16	10310	14982
17	10310	16310

Showing 1 to 18 of 28,980 entries, 2 total columns

Figure: 1.1

```
g <- graph.data.frame(d = table_as_df, directed = T)
g

```

```
IGRAPH b762768 DN-- 5242 28980 --
+ attr: name (v/c)
+ edges from b762768 (vertex names):
 [1] 3466 ->937   3466 ->5233  3466 ->8579  3466 ->10310 3466 ->15931 3466 ->17038 3466 ->18720
 [8] 3466 ->19607 10310->1854  10310->3466  10310->4583  10310->5233  10310->9572  10310->10841
[15] 10310->13056 10310->14982 10310->16310 10310->19640 10310->23855 10310->24372 10310->24814
[22] 5052 ->899   5052 ->1796  5052 ->2287  5052 ->3096  5052 ->3386  5052 ->4472  5052 ->5346
[29] 5052 ->5740  5052 ->6094  5052 ->6376  5052 ->9124  5052 ->10235 5052 ->10427 5052 ->10597
[36] 5052 ->15159 5052 ->16148 5052 ->16741 5052 ->18235 5052 ->18549 5052 ->19297 5052 ->20511
[43] 5052 ->20595 5052 ->20613 5052 ->24371 5052 ->24559 5052 ->24731 5052 ->25102 5052 ->25271
[50] 5052 ->25396 5346 ->1658  5346 ->4822  5346 ->5052  5346 ->6864  5346 ->7689  5346 ->7926
+ ... omitted several edges
```

Figure: 1.2

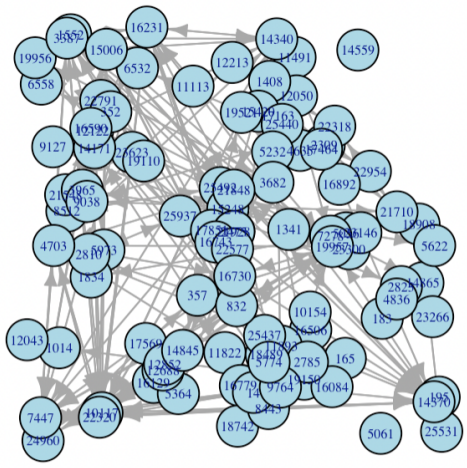


Figure: 1.3

```
library(igraph)

table <- read.table("CA-GrQc.txt", skip=3, as.is=TRUE, comment = "" , col.names = c("FromNodeId",
"ToNodeId"), header=TRUE)

table_as_df <- as.data.frame.matrix(table)

g <- graph.data.frame(d = table_as_df, directed = T)

plot(g, layout = layout_with_fr,vertex.size = 40, vertex.color = 'lightblue', edge.arrow.size =
0.1, vertex.label.cex = 0.5, edge.label = table_as_df$Relationship, edge.label.cex = 0.5)
```

Figure: 1.4

3. Applied Few Functions from Introduction to Graph Analytics Doc

E(g): Edges of the graph.

```
> View(table)
> data_frame<-as.data.frame.matrix(table)
> graph <- graph_from_data_frame(d = data_frame, directed = T)
> E(graph)
+ 28980/28980 edges from f010ed5 (vertex names):
 [1] 3466 ->937   3466 ->5233  3466 ->8579  3466 ->10310 3466 ->15931 3466 ->17038
 [7] 3466 ->18720 3466 ->19607 10310->1854  10310->3466  10310->4583  10310->5233
[13] 10310->9572  10310->10841 10310->13056 10310->14982 10310->16310 10310->19640
[19] 10310->23855 10310->24372 10310->24814 5052 ->899   5052 ->1796  5052 ->2287
[25] 5052 ->3096  5052 ->3386  5052 ->4472  5052 ->5346  5052 ->5740  5052 ->6094
[31] 5052 ->6376  5052 ->9124  5052 ->10235 5052 ->10427 5052 ->10597 5052 ->15159
[37] 5052 ->16148 5052 ->16741 5052 ->18235 5052 ->18549 5052 ->19297 5052 ->20511
[43] 5052 ->20595 5052 ->20613 5052 ->24371 5052 ->24559 5052 ->24731 5052 ->25102
[49] 5052 ->25271 5052 ->25396 5346 ->1658  5346 ->4822  5346 ->5052  5346 ->6864
[55] 5346 ->7689  5346 ->7926  5346 ->9124  5346 ->10268 5346 ->12971 5346 ->15159
+ ... omitted several edges
```

Figure: 3.1

Write-Up:

E(g) function provides all the edges present in a graph and in this case there are 23980 edges and as we can see in Figure: 3.1, we can notice few of the edges.

V(g): Vertices of the graph.

```
> V(graph)
+ 5242/5242 vertices, named, from f010ed5:
   [1] 3466  10310 5052  5346  15159 19640 10243 18648 16470 17822 14265 19738 8612  10822
  [15] 16258 21194 14123 2710  18757 16148 10794 7050  4846  824   2133  6610  6700  11082
  [29] 14419 17330 18487 22779 23382 12928 13740 13096 22393 3872  23096 8862  22598 8254
  [43] 17309 24833 10115 23916 7893  18051 4515  16778 8978  9017  15170 15455 16589 7510
  [57] 8851  624   2654  10130 16032 3937  21012 22691 19215 2980  14376 9522  9572  17394
  [71] 18924 3441  10268 7188  18866 4870  5175  8282  22046 8157  2120  7713  19052 8302
  [85] 16484 17778 21699 8701  17379 6858  8148  15366 9360  11102 25758 9755  14344 24163
  [99] 12691 3420  7442  7768  4624  15770 23099 14337 15799 11241 25271 25396 16994 21776
 [113] 3345  8868  11077 17162 21389 6895  12968 15144 19974 8708  23776 13597 20303 3032
 [127] 22415 4700  9124  4046  21806 3409  18001 22177 25480 8916  12334 13713 2949  7042
+ ... omitted several vertices
```

Figure: 3.2

Write-Up:

V(g) function provides all the nodes/ vertices present in a graph and in this case there are 5242 nodes and as we can see Figure: 3.2, we can notice few of the edges.

Edge Density: Density of the graph shows how strongly the graph is connected. As we know that density is defined as the number of edges that are existing divided by the number of possible edges. Using density we can find the connectivity of the graph.

```
> edge_density(graph)
[1] 0.001054841
```

Figure: 3.3

Write-Up:
High density means a graph is highly connected whereas low density means the graph is loosely connected. As we can see above the density of the graph is very less, that means this graph is not heavily connected. Figure 3.3 shows the edge density of the graph. The graph is not highly connected.

Degree: Degree of a node is the number of edges that are connected to that particular node.

```
> degree(graph,v= V(graph), mode = c("all"))
 3466 10310  5052  5346 15159 19640 10243 18648 16470 17822 14265 19738  8612 10822 16258
   16    26    58    40     8    50    28     8     2     2    74    10    36     4    26
21194 14123  2710 18757 16148 10794  7050  4846   824  2133  6610  6700 11082 14419 17330
    2    26    66    16    36     4    12    44    26    24   136    62    24    28    34
18487 22779 23382 12928 13740 13096 22393  3872 23096  8862 22598  8254 17309 24833 10115
   54    22    60    22    22    44    10    16     2    14    36    26     2    34     4
23916  7893 18051  4515 16778  8978  9017 15170 15455 16589  7510  8851   624  2654 10130
    2    18    20    16    16     8    56     8    34    10     4    22    34    76    30
16032  3937 21012 22691 19215  2980 14376  9522  9572 17394 18924  3441 10268  7188 18866
   32     8   162   154     4    48     2    10    68     2    16    12    14    24    88
 4870  5175  8282 22046  8157  2120  7713 19052  8302 16484 17778 21699  8701 17379  6858
    6    12     6    22     2     2     2     2     2     2    10     6    20    28     6
 8148 15366  9360 11102 25758  9755 14344 24163 12691  3420  7442  7768  4624 15770 23099
    6     6    20    10   102    62     6     4     4     6    36    20    24    10     4
14337 15799 11241 25271 25396 16994 21776  3345  8868 11077 17162 21389  6895 12968 15144
   18    18    98    32    46    40    10    30     2    28     4    10     2    20    26
19974  8708 23776 13597 20303  3032 22415  4700  9124  4046 21806  3409 18001 22177 25480
```

Figure: 3.4

Write-Up:
In Figure: 3.4, we can clearly see the degree of the initial portion of the graph.

In-Degree: In a directed graph, the in-degree of a node is the number of edges which are having that particular node as endpoint. Figure 2.5 shows the in-degree of the initial portion of the graph.

```
> degree(graph,v= V(graph), mode = c("in"))
 3466 10310  5052  5346 15159 19640 10243 18648 16470 17822 14265 19738  8612 10822 16258
    8    13    29    20     4    25    14     4     1     1    37     5    18     2    13
21194 14123  2710 18757 16148 10794  7050  4846   824  2133  6610  6700 11082 14419 17330
    1    13    33     8    18     2     6    22    13    12    68    31    12    14    17
18487 22779 23382 12928 13740 13096 22393  3872 23096  8862 22598  8254 17309 24833 10115
   27    11    30    11    11    22     5     8     1     7    18    13     1    17     2
23916  7893 18051  4515 16778  8978  9017 15170 15455 16589  7510  8851   624  2654 10130
    1     9    10     8     8     4    28     4    17     5     2    11    17    38    15
16032  3937 21012 22691 19215  2980 14376  9522  9572 17394 18924  3441 10268  7188 18866
   16     4    81    77     2    24     1     5    34     1     8     6     7    12    44
 4870  5175  8282 22046  8157  2120  7713 19052  8302 16484 17778 21699  8701 17379  6858
    3     6     3    11     1     1     1     1     1     1     5     3    10    14     3
 8148 15366  9360 11102 25758  9755 14344 24163 12691  3420  7442  7768  4624 15770 23099
    3     3    10     5    51    31     3     2     2     3    18    10    12     5     2
14337 15799 11241 25271 25396 16994 21776  3345  8868 11077 17162 21389  6895 12968 15144
    9     9    49    16    23    20     5    15     1    14     2     5     1    10    13
19974  8708 23776 13597 20303  3032 22415  4700  9124  4046 21806  3409 18001 22177 25480
```

Figure: 3.5

Write-Up:

In Figure: 3.5, we can clearly see the in-degree of the initial portion of the graph.

Out-Degree: Out-degree of node is the number of edges which are having that particular node as starting point.

```
> degree(graph,v= V(graph), mode = c("out"))
 3466 10310  5052  5346 15159 19640 10243 18648 16470 17822 14265 19738  8612 10822 16258
    8    13    29    20     4    25    14     4     1     1    37     5    18     2    13
21194 14123  2710 18757 16148 10794  7050  4846   824  2133  6610  6700 11082 14419 17330
    1    13    33     8    18     2     6    22    13    12    68    31    12    14    17
18487 22779 23382 12928 13740 13096 22393  3872 23096  8862 22598  8254 17309 24833 10115
   27    11    30    11    11    22     5     8     1     7    18    13     1    17     2
23916  7893 18051  4515 16778  8978  9017 15170 15455 16589  7510  8851   624  2654 10130
    1     9    10     8     8     4    28     4    17     5     2    11    17    38    15
16032  3937 21012 22691 19215  2980 14376  9522  9572 17394 18924  3441 10268  7188 18866
   16     4    81    77     2    24     1     5    34     1     8     6     7    12    44
 4870  5175  8282 22046  8157  2120  7713 19052  8302 16484 17778 21699  8701 17379  6858
    3     6     3    11     1     1     1     1     1     1     5     3    10    14     3
 8148 15366  9360 11102 25758  9755 14344 24163 12691  3420  7442  7768  4624 15770 23099
    3     3    10     5    51    31     3     2     2     3    18    10    12     5     2
14337 15799 11241 25271 25396 16994 21776  3345  8868 11077 17162 21389  6895 12968 15144
    9     9    49    16    23    20     5    15     1    14     2     5     1    10    13
19974  8708 23776 13597 20303  3032 22415  4700  9124  4046 21806  3409 18001 22177 25480
```

Figure: 3.6

Write-Up:

In Figure: 3.6, we can clearly see the out-degree of the initial portion of the graph.

As the graph is undirected both in-degree and out-degree are identical, We can see this in Figure: 3.7.

```
> out_degree <- degree(graph, mode = c("out"))
> in_degree <- degree(graph, mode = c("in"))
> identical(in_degree,out_degree)
[1] TRUE
```

Figure: 3.7

gorder(graph): gorder() gives the number of vertices of the graph.

```
> gorder(graph)
[1] 5242
```

Figure: 3.8

Write-Up:

From figure 3.8, we can clearly see that there are 5242 vertices in the entire graph.

gsize(graph): gsize() gives the number of edges of the graph.

```
> gsize(graph)
[1] 28980
```

Figure: 3.9

Write-Up:
From figure 3.9, we can clearly see that there are 28980 edges in the entire graph.

Simplify(graph): Simplify() is used to simplify the graph.

```
> simplify(graph)
IGRAPH 1ae49ae DN-- 5242 28968 --
+ attr: name (v/c)
+ edges from 1ae49ae (vertex names):
 [1] 3466 ->10310 3466 ->5233  3466 ->17038 3466 ->8579  3466 ->15931 3466 ->937
 [7] 3466 ->18720 3466 ->19607 10310->3466  10310->19640 10310->9572  10310->5233
[13] 10310->16310 10310->10841 10310->23855 10310->4583  10310->24372 10310->13056
[19] 10310->24814 10310->1854  10310->14982 5052 ->5346  5052 ->15159 5052 ->16148
[25] 5052 ->25271 5052 ->25396 5052 ->9124  5052 ->24371 5052 ->16741 5052 ->19297
[31] 5052 ->10235 5052 ->6094  5052 ->24559 5052 ->4472  5052 ->18549 5052 ->10597
[37] 5052 ->2287  5052 ->20595 5052 ->20613 5052 ->24731 5052 ->3096  5052 ->20511
[43] 5052 ->1796  5052 ->3386  5052 ->5740  5052 ->6376  5052 ->25102 5052 ->10427
+ ... omitted several edges
>
> gorder(graph)
[1] 5242
>
> gsize(graph)
[1] 28980
```

Figure: 3.10

Write-Up:
From figure 3.10, we can clearly see that the number of edges and vertices of the graph are still the same even after applying a simplify function that means there are no loop and multiple edges in the graph.

From the rubric, you see a plot of the first 50 collaborators. Try this the first 100 nodes and see what the result is. How does this compare to the first 50 nodes

We selected the first 50 nodes in the graph to plot the graph and to see the edges and edge_density among the nodes. We can see this in figures 3.11 & 3.12.
Next, we selected the first 100 nodes in the graph to plot the graph and to see the edges and edge_density among the nodes. We can see this in figures 3.13 & 3.14.

Plot, edges and edge density of the first 50 nodes

```
first_50 <- delete.vertices(graph,-V(graph)[1:50])
plot(first_50, vertex.size = 20, vertex.color = 'lightblue', vertex.label.cex=0.5, layout=
layout.lgl, edge.arrow.size=0.6)
```

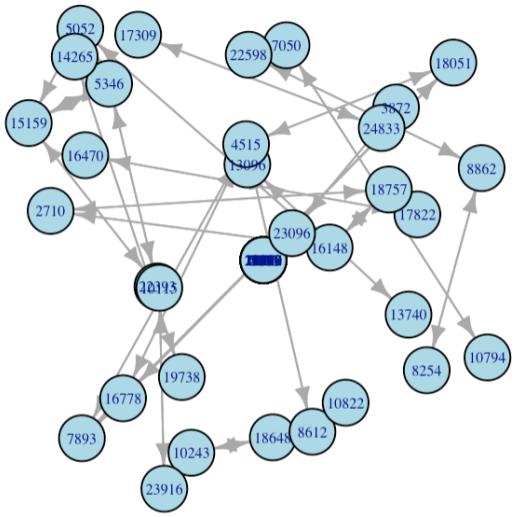


Figure: 3.11

```
> E(first_50)
+ 200/200 edges from 1acf81e (vertex names):
 [1] 3466 ->10310 10310->3466  10310->19640 5052 ->5346  5052 ->15159 5052 ->16148
 [7] 5346 ->5052  5346 ->15159 5346 ->22393 15159->5052  15159->5346  15159->22393
[13] 19640->6610  19640->6700  19640->10310 10243->18648 18648->10243 16470->17822
[19] 17822->16470 14265->19738 19738->14265 8612 ->4515  8612 ->10822 10822->8612
[25] 16258->11082 16258->14123 16258->21194 21194->16258 14123->16258 2710 ->16148
[31] 2710 ->18757 18757->2710  18757->16148 16148->2710  16148->5052  16148->18757
[37] 10794->7050  7050 ->10794 4846 ->824   4846 ->2133  4846 ->6610  4846 ->6700
[43] 4846 ->11082 4846 ->12928 4846 ->14419 4846 ->17330 4846 ->18487 4846 ->22779
[49] 4846 ->23382 824  ->2133  824  ->4846  824  ->6610  824  ->6700  824  ->11082
[55] 824  ->12928 824  ->14419 824  ->17330 824  ->18487 824  ->22779 824  ->23382
+ ... omitted several edges
> edge_density(first_50)
[1] 0.08163265
```

Figure: 3.12

Plot, edges and edge density of the first 50 nodes

```
first_100 <- delete.vertices(graph,-V(graph)[1:100])
plot(first_100, vertex.size = 20, vertex.color = 'lightblue', vertex.label.cex=0.5, layout=
layout.lgl, edge.arrow.size=0.6)
```

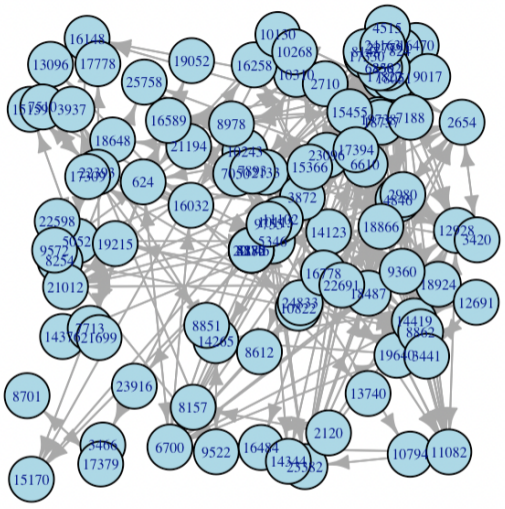


Figure: 3.13

```
> E(first_100)
+ 364/364 edges from 570a0d7 (vertex names):
 [1] 3466 ->10310 10310->3466  10310->9572  10310->19640 5052 ->5346  5052 ->15159
 [7] 5052 ->16148 5346 ->5052  5346 ->10268 5346 ->15159 5346 ->22393 15159->5052
[13] 15159->5346  15159->22393 19640->624   19640->6610  19640->6700  19640->10310
[19] 10243->18648 10243->21012 10243->22691 18648->10243 16470->17822 17822->16470
[25] 14265->3441  14265->3937  14265->9522  14265->12691 14265->19738 19738->9572
[31] 19738->14265 8612 ->4515  8612 ->10822 10822->8612  16258->11082 16258->14123
[37] 16258->21194 21194->16258 14123->16258 2710 ->16148 2710 ->18757 18757->2710
[43] 18757->16148 16148->2710  16148->5052  16148->18757 10794->7050  7050 ->10794
[49] 4846 ->824   4846 ->2133  4846 ->2654  4846 ->6610  4846 ->6700  4846 ->11082
[55] 4846 ->12928 4846 ->14419 4846 ->17330 4846 ->18487 4846 ->22779 4846 ->23382
+ ... omitted several edges
> edge_density(first_100)
[1] 0.03676768
```

Figure: 3.14

Observation: As expected, the nodes and edges will be more in the first 100 than the first 50. The edge density got reduced almost half.

Now, Let's pick the 50 nodes around the halfway node; then the 100 nodes.

We selected the 50 nodes around the halfway node in the graph to plot the graph and to see the edges and edge_density among the nodes. We can see this in Figure 3.15 & 2.16.
Next, we selected 100 nodes ar ound the halfway node in the graph to plot the graph and to see the edges and edge_density among the nodes. We can see this in Figuree 2.17 & 2.18.
From the plots we came to know that there are no nodes connected around the halfway node. As there are no edges the edge density is 0.

Plot, edges and edge density of the 50 nodes around halfway node

```
mid_50 <- induced.subgraph(graph, V(graph)[2597:2646])
plot(mid_50, vertex.size = 20, vertex.color = 'lightblue', vertex.label.cex=0.5, layout=
layout.graphopt, edge.arrow.size=0.6)

```

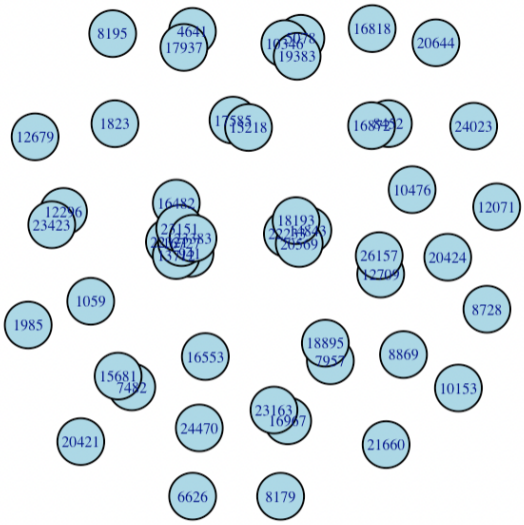


Figure: 3.15

```
> E(mid_50)
+ 66/66 edges from 2257a2d (vertex names):
 [1] 16482->23151 22321->941   22321->13712 22321->16727 22321->23151 22321->23383
 [7] 941  ->22321 941  ->13712 941  ->16727 941  ->23151 941  ->23383 13712->22321
[13] 13712->941   13712->16727 13712->23151 13712->23383 16727->22321 16727->941
[19] 16727->13712 16727->23151 16727->23383 23151->16482 23151->22321 23151->941
[25] 23151->13712 23151->16727 23151->23383 23383->22321 23383->941   23383->13712
[31] 23383->16727 23383->23151 12296->23423 23423->12296 4641 ->17937 17937->4641
[37] 7482 ->15681 15681->7482  5078 ->10346 5078 ->19383 10346->5078  10346->19383
[43] 19383->5078  19383->10346 8452 ->16872 16872->8452  12709->26157 26157->12709
[49] 22233->14843 22233->20569 22233->18193 14843->22233 14843->20569 14843->18193
[55] 20569->22233 20569->14843 20569->18193 18193->22233 18193->14843 18193->20569
+ ... omitted several edges
> edge_density(mid_50)
[1] 0.02693878
```

Figure: 3.16

Plot, edges and edge density of the 100 nodes around halfway node

```
mid_100 <- induced.subgraph(graph, V(graph)[2572:2671])
plot(mid_100, vertex.size = 20, vertex.color = 'lightblue', vertex.label.cex=0.5, layout=
layout.graphopt, edge.arrow.size=0.6)
```

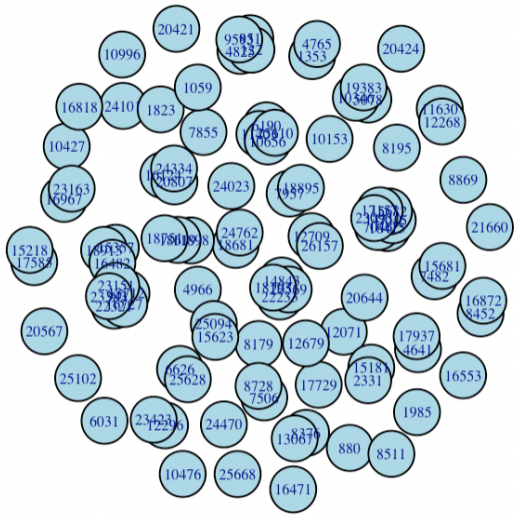


Figure: 3.17

```
> E(mid_100)
+ 164/164 edges from da7a0ad (vertex names):
 [1] 8376 ->13067 13067->8376  7506 ->8728  18681->24762 24762->18681 16124->20807 16124->24334
 [8] 20807->16124 20807->24334 24334->16124 24334->20807 16415->14710 16415->7046  16415->7045
[15] 16415->15682 16415->17157 16415->23093 14710->16415 14710->7046  14710->7045  14710->15682
[22] 14710->17157 14710->23093 7046 ->16415 7046 ->14710 7046 ->7045  7046 ->15682 7046 ->17157
[29] 7046 ->23093 7045 ->16415 7045 ->14710 7045 ->7046  7045 ->15682 7045 ->17157 7045 ->23093
[36] 15682->16415 15682->14710 15682->7046  15682->7045  15682->17157 15682->23093 17157->16415
[43] 17157->14710 17157->7046  17157->7045  17157->15682 17157->23093 23093->16415 23093->14710
[50] 23093->7046  23093->7045  23093->15682 23093->17157 15357->18913 15357->16482 18913->15357
[57] 18913->16482 16482->15357 16482->18913 16482->23151 22321->941   22321->13712 22321->16727
[64] 22321->23151 22321->23383 941  ->22321 941  ->13712 941  ->16727 941  ->23151 941  ->23383
+ ... omitted several edges
> edge_density(mid_100)
[1] 0.01656566
```

Figure: 3.18

Observation: As expected, the nodes and edges will be more in the mid 100 than the mid 50. The edge density got reduced almost half.

Now, Let's DO the last 50 nodes and the last 100 nodes.

This time we selected the last 50 nodes in the graph to plot the graph and to see the edges and edge_density among the nodes. We can see this in Figure 3.19 & 3.20.

Next, we selected the last 100 nodes in the graph to plot the graph and to see the edges and edge_density among the nodes. We can see this in Figure 3.21 & 3.22.

Plot, edges and edge density of the last 50 nodes

```
last_50 <- induced.subgraph(graph, V(graph)[5193:5242])
plot(last_50, vertex.size = 20, vertex.color = 'lightblue', vertex.label.cex=0.5, layout=
layout.graphopt, edge.arrow.size=0.6)
```

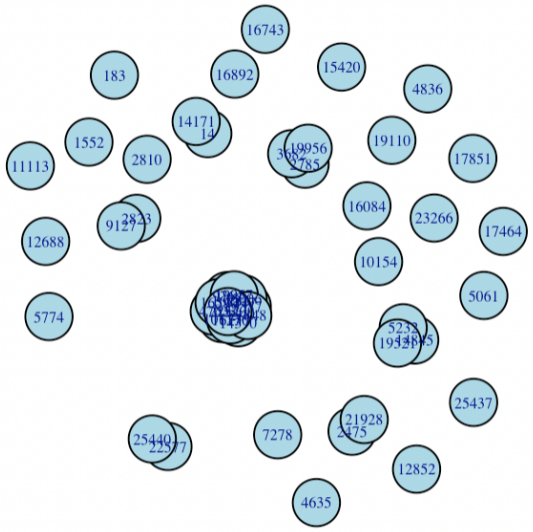


Figure: 3.19

```
> E(last_50)
+ 230/230 edges from c9ca015 (vertex names):
 [1] 2823 ->9127  9127 ->2823  2475 ->21928 21928->2475  22577->25440 25440->22577 14   ->14171
 [8] 14171->14    14845->5232  14845->19521 5232 ->14845 5232 ->19521 19521->14845 19521->5232
[15] 2785 ->3682  2785 ->19956 3682 ->2785  3682 ->19956 19956->2785  19956->3682  10117->9764
[22] 10117->4703  10117->3387  10117->832   10117->7447  10117->14340 10117->14370 10117->15006
[29] 10117->16231 10117->16590 10117->16730 10117->19957 10117->21848 10117->22320 9764 ->10117
[36] 9764 ->4703  9764 ->3387  9764 ->832   9764 ->7447  9764 ->14340 9764 ->14370 9764 ->15006
[43] 9764 ->16231 9764 ->16590 9764 ->16730 9764 ->19957 9764 ->21848 9764 ->22320 4703 ->10117
[50] 4703 ->9764  4703 ->3387  4703 ->832   4703 ->7447  4703 ->14340 4703 ->14370 4703 ->15006
[57] 4703 ->16231 4703 ->16590 4703 ->16730 4703 ->19957 4703 ->21848 4703 ->22320 3387 ->10117
[64] 3387 ->9764  3387 ->4703  3387 ->832   3387 ->7447  3387 ->14340 3387 ->14370 3387 ->15006
+ ... omitted several edges
> edge_density(last_50)
[1] 0.09387755
```

Figure: 3.20

Plot, edges and edge density of the last 100 nodes

```
last_100 <- induced.subgraph(graph, V(graph)[5143:5242])
plot(last_100, vertex.size = 20, vertex.color = 'lightblue', vertex.label.cex=0.5, layout= layout.lgl,
edge.arrow.size=0.6)
```

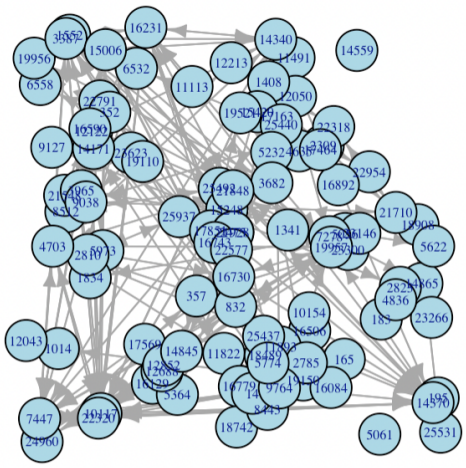


Figure: 3.21

```
> E(last_100)
+ 272/272 edges from 1461763 (vertex names):
 [1] 8512 ->25492 25492->8512  19150->1014  1014 ->19150 18908->25531 25531->18908 6558 ->14865
 [8] 14865->6558  17163->22954 22954->17163 2309 ->6532  6532 ->2309  25937->2810  5973 ->12122
[15] 12122->5973  1834 ->5056  5056 ->1834  3965 ->5622  5622 ->3965  8443 ->15248 15248->8443
[22] 357  ->23300 357  ->24960 23300->357   23300->24960 24960->357   24960->23300 9038 ->22318
[29] 9038 ->12213 9038 ->21710 22318->9038  22318->12213 22318->21710 12213->9038  12213->22318
[36] 12213->21710 21710->9038  21710->22318 21710->12213 12050->5364  5364 ->12050 2823 ->9127
[43] 9127 ->2823  2475 ->21928 21928->2475  2810 ->25937 22577->25440 25440->22577 14   ->14171
[50] 14171->14    14845->5232  14845->19521 5232 ->14845 5232 ->19521 19521->14845 19521->5232
[57] 2785 ->3682  2785 ->19956 3682 ->2785  3682 ->19956 19956->2785  19956->3682  10117->9764
[64] 10117->4703  10117->3387  10117->832   10117->7447  10117->14340 10117->14370 10117->15006
+ ... omitted several edges
> edge_density(last_100)
[1] 0.02747475
```

Figure: 3.22

Observation: Observation: As expected, the nodes and edges will be more in the last 100 than the last 50.

4. Application of Miscellaneous Functions

1. **Biconnected.components(graph):** Takes in the graph as input. Although our graph g is directed, this function will read it as an undirected graph. For reference, our graph g is biconnected if the removal of any vertex within the graph does not make it disconnected. This function gives the biconnected components of the graph; these are maximal biconnected subgraphs of it. In this particular example, we used the return value of the function to generate the size of the largest biconnected component in graph g.

```
bc <- biconnected.components(g)

print(paste("Size of Largest Biconnected Component: ", max(sapply(bc$components, length))))
```

```
[1] "Size of Largest Biconnected Component:  2651"
```

2. **Cocitation(graph):** Two vertices are cocited if there is another vertex that cites them both. It is commonly used in citation graphs to describe the relationship of citations within a collection of documents. This particular function counts how many pairs of vertices are cocited. The return is a numeric matrix that contains the cocitation for vertices v[i] and j.

```
print(cocitation(g))
```

```
3466 10310 5052 5346 15159 19640 10243 18648 16470 17822 14265 19738 8612 10822 16258 21194
14123 2710 18757 16148 10794 7050 4846 824 2133 6610 6700 11082 14419 17330 18487 22779
23382 12928 13740 13096 22393 3872 23096 8862 22598 8254 17309 24833 10115 23916 7893 18051
4515 16778 8978 9017 15170 15455 16589 7510 8851 624 2654 10130 16032 3937 21012 22691 19215
2980 14376 9522 9572 17394 18924 3441 10268 7188 18866 4870 5175 8282 22046 8157 2120 7713
19052 8302 16484 17778 21699 8701 17379 6858 8148 15366 9360 11102 25758 9755 14344 24163
12691 3420 7442 7768 4624 15770 23099 14337 15799 11241 25271 25396 16994 21776 3345 8868
11077 17162 21389 6895 12968 15144 19974 8708 23776 13597 20303 3032 22415 4700 9124 4046
21806 3409 18001 22177 25480 8916 12334 13713 2949 7042 10113 11661 13621 25388 284 6427
1044 5809 811 15123 16083 2127 11712 3412 10762 13847 17207 7689 18457 17228 20207 17559
19206 13813 7504 12212 1545 10791 10919 10588 5729 1310 23186 25931 9710 7007 17536 22476
5233 17932 1000 23107 23939 5464 13653 26038 23576 580 1877 2042 15880 16101 20014 2846
18375 8335 13529 6494 6857 24781 14308 20787 23689 25143 22184 5597 23836 6355 7194 20960
23066 14500 3765 1508 8279 2004 21191 21663 14020 16543 16563 238 8887 6072 491 1818 17082
17039 3547 14707 14818 6804 19624 214 6512 10590 18904 2190 11801 23441 449 10358 10615
12074 3099 4254 9337 1497 6724 2558 15538 21910 8968 12260 4241 24924 4697 22082 13276 15409
639 1153 17501 16834 570 19657 26190 1727 19992 9417 24009 15961 14316 14690 13008 4550
13142 24330 15666 15205 24640 15415 8365 17075 23665 78 140 24122 18365 1248 16331 17843
4952 5060 20660 11447 7383 21432 13929 5136 13556 6934 20683 25419 21771 22644 10467 13175
21816 11640 14371 17443 16310 19517 22989 23708 5934 15066 16469 4383 13614 17245 10620 1880
15353 21048 8932 6709 18003 12135 3209 6863 24371 1833 12107 8177 14967 16741 19233 19297
20597 7542 7350 7712 19149 11275 14638 11566 11808 18560 13282 15249 15624 4766 12665 3310
7449 22778 23351 23943 17754 25674 16123 10841 1403 2368 20168 8503 18719 18867 23409 10942
23647 22119 3953 5672 3430 24431 16020 23986 16675 13859 16091 2081 1941 15846 2054 10355
```

3. **Bibcoupling(graph):** Calculates the bibcoupling coupling for all vertices in our graph g. The bibcoupling of a pair of vertices equates to the number of other vertices that the pair cites. The return value is the same as the cocitation(graph) function except obviously containing the bibcoupling for vertices v[i] and j. We found there to be no examples of bibcoupling.

```
print(bibcoupling(g))
```

```
3466 10310 5052 5346 15159 19640 10243 18648 16470 17822 14265 19738 8612 10822 16258 21194
14123 2710 18757 16148 10794 7050 4846 824 2133 6610 6700 11082 14419 17330 18487 22779
23382 12928 13740 13096 22393 3872 23096 8862 22598 8254 17309 24833 10115 23916 7893 18051
4515 16778 8978 9017 15170 15455 16589 7510 8851 624 2654 10130 16032 3937 21012 22691 19215
2980 14376 9522 9572 17394 18924 3441 10268 7188 18866 4870 5175 8282 22046 8157 2120 7713
19052 8302 16484 17778 21699 8701 17379 6858 8148 15366 9360 11102 25758 9755 14344 24163
12691 3420 7442 7768 4624 15770 23099 14337 15799 11241 25271 25396 16994 21776 3345 8868
11077 17162 21389 6895 12968 15144 19974 8708 23776 13597 20303 3032 22415 4700 9124 4046
21806 3409 18001 22177 25480 8916 12334 13713 2949 7042 10113 11661 13621 25388 284 6427
1044 5809 811 15123 16083 2127 11712 3412 10762 13847 17207 7689 18457 17228 20207 17559
19206 13813 7504 12212 1545 10791 10919 10588 5729 1310 23186 25931 9710 7007 17536 22476
5233 17932 1000 23107 23939 5464 13653 26038 23576 580 1877 2042 15880 16101 20014 2846
18375 8335 13529 6494 6857 24781 14308 20787 23689 25143 22184 5597 23836 6355 7194 20960
23066 14500 3765 1508 8279 2004 21191 21663 14020 16543 16563 238 8887 6072 491 1818 17082
17039 3547 14707 14818 6804 19624 214 6512 10590 18904 2190 11801 23441 449 10358 10615
12074 3099 4254 9337 1497 6724 2558 15538 21910 8968 12260 4241 24924 4697 22082 13276 15409
639 1153 17501 16834 570 19657 26190 1727 19992 9417 24009 15961 14316 14690 13008 4550
13142 24330 15666 15205 24640 15415 8365 17075 23665 78 140 24122 18365 1248 16331 17843
4952 5060 20660 11447 7383 21432 13929 5136 13556 6934 20683 25419 21771 22644 10467 13175
21816 11640 14371 17443 16310 19517 22989 23708 5934 15066 16469 4383 13614 17245 10620 1880
15353 21048 8932 6709 18003 12135 3209 6863 24371 1833 12107 8177 14967 16741 19233 19297
20597 7542 7350 7712 19149 11275 14638 11566 11808 18560 13282 15249 15624 4766 12665 3310
7449 22778 23351 23943 17754 25674 16123 10841 1403 2368 20168 8503 18719 18867 23409 10942
23647 22119 3953 5672 3430 24431 16020 23986 16675 13859 16091 2081 1941 15846 2054 10355
```

4. **count_components(graph):** Returns the number of maximal (weakly or strongly) connected components of a graph. According to this function, only one connected component was found in the entire graph.

```
print(count_components(g))
```

```
[1] 355
```

5. **Closeness Centrality:** Clearly the closeness centrality value of almost all nodes is very small as this is not a highly connected graph.

```
centr_clo(g)
```

```
$res
   [1] 0.0009188371 0.0009192121 0.0009194177 0.0009191841 0.0009188977 0.0009192739 0.0009190875
   [8] 0.0009185556 0.0001908033 0.0001908033 0.0009192089 0.0009191937 0.0009190986 0.0009184294
  [15] 0.0009187052 0.0009180364 0.0009187017 0.0009193172 0.0009189354 0.0009193888 0.0009176185
  [22] 0.0009182865 0.0009192723 0.0009191518 0.0009190562 0.0009194032 0.0009193930 0.0009190661
  [29] 0.0009190914 0.0009191844 0.0009193442 0.0009190561 0.0009193519 0.0009190561 0.0009190804
  [36] 0.0009194027 0.0009189568 0.0009184140 0.0009177456 0.0009187269 0.0009191942 0.0009188171
  [43] 0.0009179668 0.0009186355 0.0001908397 0.0001908397 0.0009186514 0.0009186516 0.0009186397
  [50] 0.0009181396 0.0009182759 0.0009188505 0.0009182759 0.0009187933 0.0009182762 0.0009183232
  [57] 0.0009189923 0.0009192911 0.0009194821 0.0009192663 0.0009192348 0.0009188216 0.0009194732
  [64] 0.0009194456 0.0009187759 0.0009191663 0.0009184969 0.0009189596 0.0009195108 0.0009180509
  [71] 0.0009187197 0.0009186493 0.0009186884 0.0009190751 0.0009194101 0.0009183153 0.0009186395
  [78] 0.0009183153 0.0009188759 0.0001908033 0.0001908033 0.0001908033 0.0001908033 0.0001908033
  [85] 0.0001908033 0.0009188634 0.0009186699 0.0009188134 0.0009188608 0.0009179671 0.0009179671
  [92] 0.0009179671 0.0009190110 0.0009187962 0.0009193427 0.0009192911 0.0009188690 0.0009186696
  [99] 0.0009185701 0.0009189702 0.0009192255 0.0009189718 0.0009190193 0.0009183515 0.0009183509
 [106] 0.0009181102 0.0009183971 0.0009193058 0.0009192727 0.0009192943 0.0009193040 0.0009190372
 [113] 0.0009186632 0.0009179944 0.0009184220 0.0009177538 0.0009177543 0.0009183087 0.0009189779
 [120] 0.0009189062 0.0009186983 0.0001908033 0.0001908033 0.0009191565 0.0009187590 0.0009191173
 [127] 0.0009191173 0.0009184051 0.0009190743 0.0009187334 0.0009181702 0.0009190654 0.0009185561
 [134] 0.0009192042 0.0009186488 0.0009191284 0.0009190790 0.0009186353 0.0009191936 0.0009184410
 [141] 0.0009187513 0.0009189692 0.0001908033 0.0001908033 0.0009173337 0.0009180015 0.0009186493
```

```
$centralization
[1] 0.0001517938

$theoretical_max
[1] 5240
```

6. **Diameter of the Graph:** The diameter of the graph is the maximum distance between the pair of vertices. Since the diameter is 17. This shows that even TGT's are voting to some other nodes. The originally extracted graph has a bulk of frontier nodes which makes it difficult to interpret and analyze the interior nodes. So, the extracted data needs to be further processed/simplified.

```
g.adj = get.adjacency(g)
g.adj = graph_from_adjacency_matrix(g.adj)
g.dia = diameter(g.adj)
g.dia
```

```
[1] 17
```

7. **count_triangles(graph):** In an effort to figure out how many triangles are in our larger Graph g, we decided to resort to the count_triangles() function. The return value is a numeric vector that contains the number of triangles for each vertex in the graph. The plotted graph is identical to the input graph except that each vertex is labeled with the number of adjacent triangles to that particular vertex. As you can see in our Graph g, using count_triangles(g) and the sum() function, we found 144780 triangles. The code along with the output can be found in Figure 4.1. The plot of Graph g can be found in Figure 4.2, however due to the size of the graph it is unreadable.

```
adjacent <- count_triangles(g)
print(sum(adjacent))
plot(g, vertex.label=adjacent)
```


```
[1] 144780
```

Figure: 4.1

```
adjacent <- count_triangles(g)
print(sum(adjacent))
plot(g, vertex.label=adjacent)
```

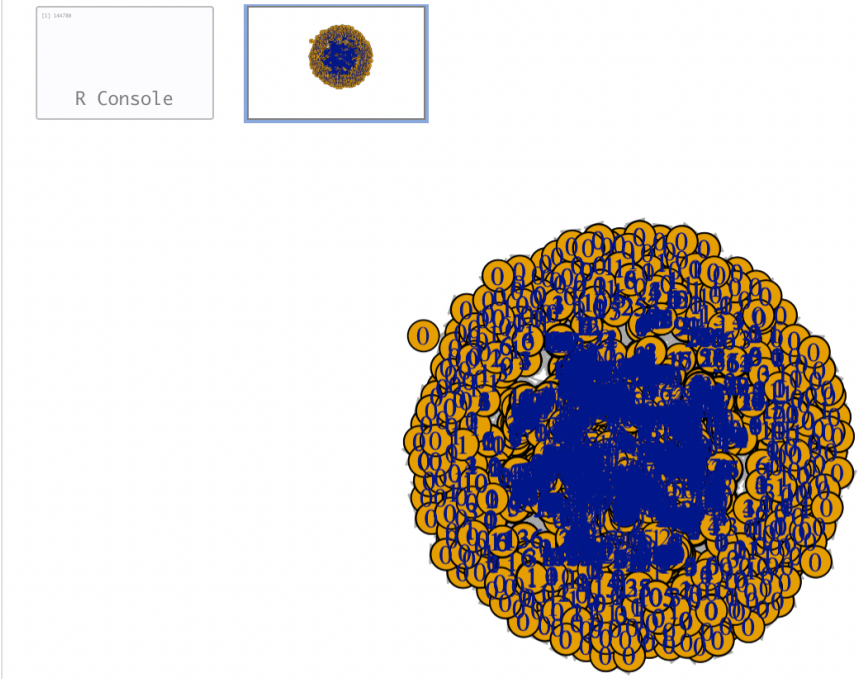


Figure: 4.2

8. **Articulation Points:** Articulation points are vertices whose removal disconnects the graph into multiple connected components. In this graph there are no articulation points which means even if we remove any vertex, still the graph is connected.

```
articulation_points(g)
```

```
+ 813/5242 vertices, named, from 9aedddc:
  [1] 16258 22483 15784 8612  8069  15123 8851  3872  2846  19624 6804  3524  10555 21613 12491
 [16] 7504  9124  5464  3345  2080  5385  8335  12135 24765 14599 15770 4624  12968 7050  1497
 [31] 6724  24183 23576 2420  19350 11861 23394 10600 23665 3754  17918 23066 3765  930   25931
 [46] 4798  6583  19697 7277  18408 6179  1727  24559 5488  10719 11032 22074 21177 21830 17245
 [61] 17443 6503  21695 21184 1711  13859 11052 12587 24402 11696 21543 15305 20236 11902 15568
 [76] 3953  22119 10456 16469 11712 11112 14864 6556  3804  1787  13056 14316 238   25469 15951
 [91] 8195  21297 11194 17501 20248 16834 20660 140   23986 5194  11066 10445 16765 11077 5225
[106] 3849  7010  24097 21771 7584  8678  11806 5787  21927 25069 2239  18924 8312  25902 9082
[121] 23896 9306  24614 20030 6337  24814 3316  21650 22502 12659 25125 12286 5993  23224 10081
[136] 8666  3136  21943 24009 18297 24833 20618 15357 10726 15395 17297 6280  19800 13673 12318
+ ... omitted several vertices
```

9. **Minimum Spanning Tree:** In order to calculate the minimum spanning tree(s) of the graph (which is a subgraph of a connected graph and the sum of the edge weights are the minimal among all subgraphs in the larger graph), this function was used. The return value of the function is a graph object.
   For the weights, the relationship column in the table_as_df variable was used. On the large Graph g, you can see in Figure: 4.3 that it looks like a blue circle. This is due to how many nodes are in the graph.
   In Figure: 4.4, both the returned graph object and the error message for the distance is shown. The error message is due to space limitations of our personal computer.
   The goal was to calculate the distance of the MST using the distance(graph) function. The returned graph object is pretty self explanatory as it gives the edge and vertex attributes. Figure: 4.5 features the code that was used to generate our results.

```
plot(mst(g, weights = table_as_df$Relationship))
```

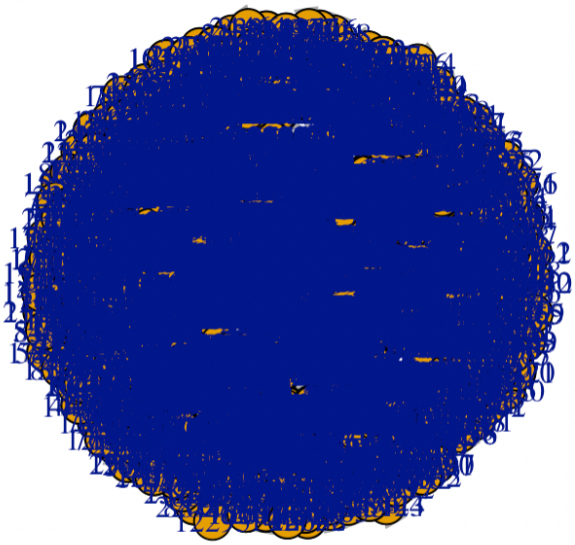


Figure: 4.3

```
> mst(g, weights = table_as_df$Relationship)
IGRAPH 2cb8647 DN-- 5242 4887 --
+ attr: name (v/c)
+ edges from 2cb8647 (vertex names):
 [1] 3466 ->937   3466 ->5233  3466 ->8579  3466 ->10310 3466 ->15931 3466 ->17038 3466 ->18720
 [8] 3466 ->19607 10310->1854  10310->4583  10310->9572  10310->10841 10310->13056 10310->14982
[15] 10310->16310 10310->19640 10310->23855 10310->24372 10310->24814 5052 ->899   5052 ->1796
[22] 5052 ->3386  5052 ->5740  5052 ->6094  5052 ->10597 5052 ->16741 5052 ->18235 5052 ->18549
[29] 5052 ->24731 5346 ->1658  5346 ->4822  5346 ->6864  5346 ->9124  5346 ->10268 5346 ->12971
[36] 5346 ->15159 5346 ->18600 5346 ->20421 5346 ->20886 5346 ->21048 5346 ->23186 5346 ->23214
[43] 5346 ->23298 5346 ->23945 5346 ->24939 19640->339   19640->624   19640->3731  19640->4743
[50] 19640->5407  19640->6610  19640->6700  19640->8045  19640->9099  19640->9639  19640->9785
+ ... omitted several edges
```

Figure: 4.4

```
distances(mst(g, weights = table_as_df$Relationship))
```

```
3466 10310 5052 5346 15159 19640 10243 18648 16470 17822 14265 19738 8612 10822 16258 21194
14123 2710 18757 16148 10794 7050 4846 824 2133 6610 6700 11082 14419 17330 18487 22779
23382 12928 13740 13096 22393 3872 23096 8862 22598 8254 17309 24833 10115 23916 7893 18051
4515 16778 8978 9017 15170 15455 16589 7510 8851 624 2654 10130 16032 3937 21012 22691 19215
2980 14376 9522 9572 17394 18924 3441 10268 7188 18866 4870 5175 8282 22046 8157 2120 7713
19052 8302 16484 17778 21699 8701 17379 6858 8148 15366 9360 11102 25758 9755 14344 24163
12691 3420 7442 7768 4624 15770 23099 14337 15799 11241 25271 25396 16994 21776 3345 8868
11077 17162 21389 6895 12968 15144 19974 8708 23776 13597 20303 3032 22415 4700 9124 4046
21806 3409 18001 22177 25480 8916 12334 13713 2949 7042 10113 11661 13621 25388 284 6427
1044 5809 811 15123 16083 2127 11712 3412 10762 13847 17207 7689 18457 17228 20207 17559
19206 13813 7504 12212 1545 10791 10919 10588 5729 1310 23186 25931 9710 7007 17536 22476
5233 17932 1000 23107 23939 5464 13653 26038 23576 580 1877 2042 15880 16101 20014 2846
18375 8335 13529 6494 6857 24781 14308 20787 23689 25143 22184 5597 23836 6355 7194 20960
23066 14500 3765 1508 8279 2004 21191 21663 14020 16543 16563 238 8887 6072 491 1818 17082
17039 3547 14707 14818 6804 19624 214 6512 10590 18904 2190 11801 23441 449 10358 10615
12074 3099 4254 9337 1497 6724 2558 15538 21910 8968 12260 4241 24924 4697 22082 13276 15409
639 1153 17501 16834 570 19657 26190 1727 19992 9417 24009 15961 14316 14690 13008 4550
13142 24330 15666 15205 24640 15415 8365 17075 23665  78 140 24122 18365 1248 16331 17843
4952 5060 20660 11447 7383 21432 13929 5136 13556 6934 20683 25419 21771 22644 10467 13175
21816 11640 14371 17443 16310 19517 22989 23708 5934 15066 16469 4383 13614 17245 10620 1880
15353 21048 8932 6709 18003 12135 3209 6863 24371 1833 12107 8177 14967 16741 19233 19297
```

Figure: 4.5

10. **mean_distance():** It calculates the mean distance by taking the distance matrix (which contains the length of all the shortest paths from or to the other vertices in the network) and averages them. As you can see in Figure: 4.6, the code is relatively simple and the mean_distance for Graph g is 6.048515.

```
mean_distance(g,directed = TRUE)
```

```
[1] 6.048515
```

Figure: 4.6

5. Central Nodes(s), longest path(s), largest clique(s), ego(s), and power centrality

Central Nodes(s): centr_degree() is used to determine the central scores of the nodes. The node with max score will be the central node.

```
V(graph)$central_degree <- centr_degree(graph)$res

V(graph)$name[V(graph)$central_degree == max(centr_degree(graph)$res)]
```

```
[1] "21012"
```

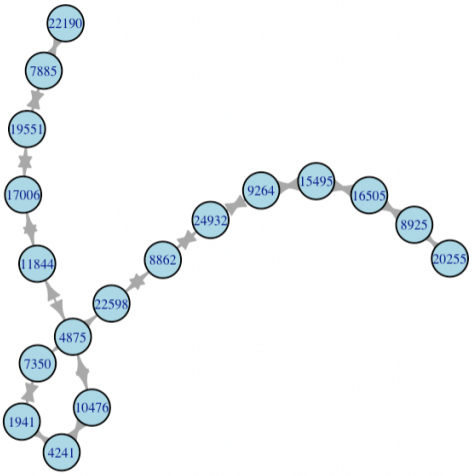
Longest Path(s): Diameter is the longest path in a graph, so here we are using get_diameter() to find the longest path/diameter of the graph.

```
diameter <- get_diameter(graph)
diameter
```

```
+ 18/5242 vertices, named, from f010ed5:
 [1] 20255 8925  16505 15495 9264  24932 8862  22598 7350  1941  4241  10476 4875  11844 17006
[16] 19551 7885  22190
```

```
longesth_path <- induced_subgraph(graph, diameter)

plot(longesth_path, vertex.size = 17, vertex.color = 'lightblue', vertex.label.cex=0.5, layout=
layout_with_graphopt, edge.arrow.size=0.6)
```



Largest Clique(s): The largest cliques in a graph can be calculated using largest_cliques() function. Figure 5.1 shows this graph contains the largest cliques each of size 44. We have plotted this clique and it contains 44 vertices and 1892 edges which we can see in Figure 5.1, 5.2 & 5.3.

```
lc <- igraph::largest_cliques(graph)
lc
```

```
Warning in igraph::largest_cliques(graph) :
  At cliques.c:1125 :directionality of edges is ignored for directed graphs
[[1]]
+ 44/5242 vertices, named, from f010ed5:
 [1] 21012 22691 773   14807 3372  21847 2741  24955 6610  25758 11241 570   6179  45    21281
[16] 23293 15003 20635 19423 18894 4164  7956  12365 17655 25346 1653  9785  21508 14540 12781
[31] 2212  19961 2952  6830  8879  11472 12496 12851 15659 17692 20108 20562 22887 4513
```

Figure: 5.1

```
lc_graph1 <- induced.subgraph(graph, lc[[1]])
lc_graph1
```

```
IGRAPH 9ecfb5d DN-- 44 1892 --
+ attr: name (v/c), central_degree (v/n)
+ edges from 9ecfb5d (vertex names):
 [1] 6610 ->21012 6610 ->22691 6610 ->25758 6610 ->11241 6610 ->570   6610 ->773   6610 ->21847
 [8] 6610 ->6179  6610 ->2741  6610 ->14807 6610 ->24955 6610 ->45    6610 ->21281 6610 ->23293
[15] 6610 ->15003 6610 ->20635 6610 ->19423 6610 ->3372  6610 ->18894 6610 ->4164  6610 ->7956
[22] 6610 ->12365 6610 ->17655 6610 ->25346 6610 ->1653  6610 ->9785  6610 ->21508 6610 ->14540
[29] 6610 ->12781 6610 ->2212  6610 ->19961 6610 ->2952  6610 ->6830  6610 ->8879  6610 ->11472
[36] 6610 ->12496 6610 ->12851 6610 ->15659 6610 ->17692 6610 ->20108 6610 ->20562 6610 ->22887
[43] 6610 ->4513  21012->6610  21012->22691 21012->25758 21012->11241 21012->570   21012->773
[50] 21012->21847 21012->6179  21012->2741  21012->14807 21012->24955 21012->45    21012->21281
+ ... omitted several edges
```

Figure: 5.2

```
plot(lc_graph1, vertex.size = 20, vertex.color = 'lightblue', vertex.label.cex=0.5, layout=
layout.graphopt, edge.arrow.size=0.5)
```

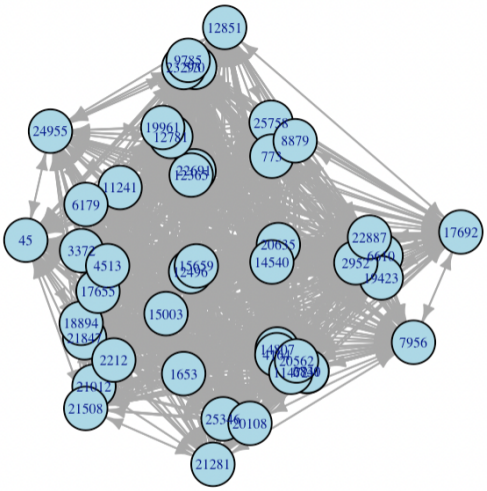


Figure: 5.3

Ego(s): In the Figure 5.4, using the Ego function, we find out the neighbors of the vertices with the given order parameter (default Order=1). In the result below, we can see the number of neighbors for each node.

```
ego_graph<-igraph::ego(graph)
ego_graph
```

```
[[1]]
+ 9/5242 vertices, named, from f010ed5:
[1] 3466  10310 5233  17038 8579  15931 937   18720 19607

[[2]]
+ 14/5242 vertices, named, from f010ed5:
 [1] 10310 3466  19640 9572  5233  16310 10841 23855 4583  24372 13056 24814 1854  14982

[[3]]
+ 30/5242 vertices, named, from f010ed5:
 [1] 5052  5346  15159 16148 25271 25396 9124  24371 16741 19297 10235 6094  24559 4472  18549
[16] 10597 2287  20595 20613 24731 3096  20511 1796  3386  5740  6376  25102 10427 18235 899

[[4]]
+ 21/5242 vertices, named, from f010ed5:
 [1] 5346  5052  15159 22393 10268 9124  7689  23186 21048 20886 4822  18600 23298 7926  1658
[16] 23945 12971 20421 24939 6864  23214
```

Figure: 5.4

Power Centrality: Both Centrality and Power are a function of the connections of the node in a neighborhood. Centrality is high if the node has more connections. Bonacich power measure corresponds to the notion that the power of a node is recursively defined by the sum of the power of its alters. The nature of the recursion involved is then controlled by the power exponent (positive or negative). Power_Centrality function finds out Bonacich power centrality scores. Here we are using exponent = 0.9 to avoid the out of memory error.

```
power_centrality(graph, exponent=0.9)
```

```
       3466        10310         5052         5346        15159        19640        10243
 0.038222990 -0.179313489  0.754899187  0.070063277  3.007790014 -1.060990120 -0.027583047
      18648        16470        17822        14265        19738         8612        10822
 3.379293307  0.479299648  0.479299648 -0.163682550 -0.828708225  0.361661557  4.213553307
      16258        21194        14123         2710        18757        16148        10794
-0.552073987 -0.448936623  0.229783643  0.963572603 -0.869736608 -2.454095938 -0.597393645
       7050         4846          824         2133         6610         6700        11082
-0.172888105 -1.030816452  0.733276150  1.195101474 -1.591023079  0.516549245  0.862593025
      14419        17330        18487        22779        23382        12928        13740
 0.071124553  0.354442324 -1.027006120  1.098875458 -0.547954526  1.098875458 -0.178649526
      13096        22393         3872        23096         8862        22598         8254
 1.203956285 -0.334833702 -1.979144709 -1.733300273  0.211281032 -0.164550303  0.345983769
      17309        24833        10115        23916         7893        18051         4515
-0.013812153 -0.068602354 -0.293764300 -0.216457906 -0.264896568  1.345735221  0.042071167
      16778         8978         9017        15170        15455        16589         7510
-0.175172366  0.108121823  0.455209386  0.108121823 -1.001187487  0.344969571  0.489552156
       8851          624         2654        10130        16032         3937        21012
-0.052116349 -0.504826205  0.182902721 -0.544105160  1.249151307  0.438709336  1.249606436
      22691        19215         2980        14376         9522         9572        17394
```

List of Functions:

- read.table()
- as.data.frame()
- graph.from.data.frame()
- plot()
- edge_density()
- igraph::E()
- igraph::V()
- degree()
- gorder()
- gsize()
- simplify()
- delete.vertices()
- induced.subgraph()
- biconnected.components()
- cocitation()
- bibcoupling()
- count_components()
- centr_clo()
- get.adjacency()
- graph_from_adjacency_matrix()
- diameter()

- count_triangles()
- articulation_points()
- mst()
- get_diameter()
- mean_distance()
- igraph::largest_cliques()
- igraph::ego()
- power_centrality()